モデル形成支援のための仕様記述変換技術

張 漢明[†,††] 荒木 啓二郎[†,†††]

本研究の目的は，ソフトウェア開発の仕様記述の工程におけるモデル化の作業を支援することである．仕様記述の過程では，システムに対する理解が深まるにつれて，より最適なモデルを用いて仕様は書き換えられる．本研究では，モデル間の仕様記述の変換手法を提示することにより，対象システムのモデル化支援を図る．本論文では，形式仕様記述言語Zを用いて2項関係モデル間の対応を分析し，2項関係モデル間の系統的な仕様記述変換の手法を提示する．仕様記述変換技術は，仕様記述の内容を維持しつつモデルを相互に変換する手法を与えることにより，目的に応じた最適なモデルの選択を可能とし，また，仕様を記述する時の指針となり得ることが期待できる．

Specification Transformation Techniques to Support Modeling for an Application System

HAN-MYUNG CHANG[†,††] and KEIJIRO ARAKI[†,†††]

One of the most important issues in software development is how to make a mathematical model for an application system. In this paper we present transformation methods for specifications in Z, which support building a formal model for an application system. We discuss relationships between transformed specifications in binary relation models such as total functions, partial functions or relations. The development of specification transformation methods means to clear and formalize the specifier's process of modeling and specification. We expect that reliability for the specification increases because of examining the specification with an appropriate model for the system.

1. はじめに

ソフトウェア開発における重要な課題として，ソフトウェア開発の上流工程である仕様記述の工程をどのように支援するかが挙げられる．仕様記述の工程では，どのようなモデルで開発対象のシステムを記述するかが最も重要である．モデルの良否がシステム全体の良否を決定するといっても過言ではない．しかし，システムを記述するための最適なモデルは最初から得られるわけではない．仕様記述者が仕様記述の過程を通して，システムに対する理解が深まるにつれて，より最適なモデルを用いて仕様は書き換えられる．本研究では，モデル間の関係を明確にすることにより，モデル間の仕様記述変換手法を提示し，対象システムのモデル化の支援を図る．

本研究の目的は，ソフトウェア開発の仕様記述の工程におけるモデル化の作業を支援することである．仕様記述者は自由な発想で対象システムの仕様記述を試みる．仕様を記述する過程で，仕様記述者は無意識のうちにシステムの分析を行い，対象システムに対する理解を深める．仕様記述の過程では，システム分析の結果，より最適な仕様記述を得るために仕様記述を再構築する．仕様記述の再構築の要因としては，データ構造の変更や仕様記述の構造の変更などが考えられる．本研究では，このようなモデルの変更による仕様記述の再構築の過程に着目した．仕様記述の再構築を仕様記述変換と捉え，再構築の過程を系統的に行う手法を確立することを目指している．

仕様記述の変換を形式的に扱うために形式的手法を導入し，モデル間の対応を形式的に表現することにより，仕様記述の変換を系統的に行う手法を提示する．形式的手法とは，コンピュータシステムのモデル化，設計，解析をおこなうための数学をベースとした技術である．近年，実際のソフトウェア開発に対して，形式的手法の適用が盛んに試みられ，文献1)，2)では，形式的手法を用いた実際の開発プロジェクトの事

† 財団法人九州システム情報技術研究所
Institute of Systems & Information Technologies
†† 奈良先端科学技術大学院大学
Nara Institute of Science and Technology
††† 九州大学
Kyushu University

例に対する分析と評価が報告されている．また，ソフトウェア開発における形式的手法の有用性については文献3)において明解に述べられている．

本論文では，形式仕様記述言語Z[4]を用いて2項関係モデル間の対応を分析し，2項関係モデル間の形式的な仕様記述変換の手法について述べる．Zにおいて2項関係モデルの中で最も特徴的なモデルである，全域関数，部分関数および関係の間の対応関係を明確にし，全域関数と部分関数の間の変換手法，および部分関数と関係の間の変換手法をそれぞれ提示する．全域関数は，定義域において未定義項を考慮する必要がないので，対象システムの概念の本質を端的に表現し易いモデルであると考えられるが，実装を考慮すれば部分関数および関係によるモデル化が有効である場合が多い．本変換手法を用いることにより，まず，考え易い全域関数のモデルで仕様記述を検討した後，部分関数化さらに関係化という過程をとることにより，プログラムに近い仕様記述を系統的に構築することを可能にする．

本変換技術の特徴は，「仕様記述の内容を維持しつつモデルを相互に変換する手法を与えること」である．変換の作業では，仕様の意味に関係なく記述を記号処理として形式的にできる変換と，仕様記述者が仕様の意味を解釈する必要がある変換がある．変換手法の提示は，形式的な変換作業と，意味を考慮する必要がある変換作業の分離を明示する．仕様記述変換は仕様記述者に対して，モデルを変換する時に何を考えればよいかを明示することにより，なぜモデルを変更するかの意義を明確にするとともに，それが仕様記述を構築する時の指針となることが期待できる．また，記号処理として形式的に行うことができる変換作業は，機械による支援ツールを開発することにより，書き換えによるエラーの混入を防ぐことが期待できる．変換技術の蓄積は，機械にできることは機械に任せ，人間が仕様記述を構築する上で本質的なシステムの意味を考えるための開発環境の基礎づけを与える．

本論文の構成は，第2章において基本概念として，形式仕様記述言語Zの概要と仕様記述変換の概念を説明する．次に，第3章では，Zにおける2項関係モデルとして，全域関数，部分関数および関係の間の対応関係を明示し，それぞれのモデル間の仕様記述の変換手法を提示する．第4章では，具体例として予約管理システムの仕様記述を用いて仕様記述変換の適用例を示す．最後に第5章では，仕様記述変換の有効性について議論する．なお，本論文におけるZの表記は文献4)による表記法を使用した．

2. 基本概念

本章では，形式仕様記述言語Zの概要と仕様記述変換の概念について述べる．

2.1 Zの概要

Z[4)~10)]は集合論と一階述語論理を基にしたモデル指向の形式仕様記述言語である．Zは1970年代後半から英国のオックスフォード大学のPRG（Programming Research Group）を中心に開発された．Zの仕様記述では，関係や関数などの数学モデルを用いて対象システムの記述を行い，システムが満たすべき性質を簡潔かつ厳密に記述する．Zでは，対象システムが満たすべき性質を記述することを目的としており，システムが「どのように」振舞うかということではなく，システムが「何」をすべきかについて着目して仕様記述を行う．

一般的なZの仕様記述では，対象システムを

- システムの内部状態，と
- システムの操作

を定義することにより表現する．システムが保持すべきシステムの内部状態は，集合，関係，関数などの概念を用いてモデル化される．また，システムの動的振舞いを表現する操作は，入力，出力，および実行前後のシステム内部状態の間の関係を用いてモデル化される．

一般的なZ仕様記述の構成について，簡単な例として，ある団体の会員の名前を保持するシステムの仕様記述を示す．Zにおける最も特徴的なものとして，スキーマによる仕様記述の図式表現が挙げられる．システム内部状態と操作の仕様記述は，それぞれ状態スキーマおよび操作スキーマを用いて定義する．

Zでは，基本的なデータの型として，システムが定義している基本型（basic type）と，ユーザが定義する型（given set）がある．given setではその内部構造については言及しない．

$[PERSON]$

$PERSON$ は人の名前の集合を表す．

システム内部状態として，会員の名前を保持するためのシステム内部状態を定義する．

Member

$$member : \mathbb{P}\, PERSON$$

$$\#member < 100$$

Member はスキーマの名前を表している．スキーマの表記は，宣言部（上半分）と述語部（下半分）に分か

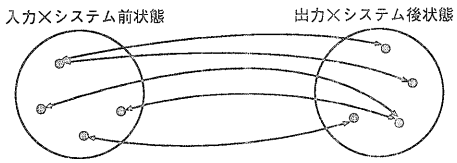

図1 操作の仕様記述の概念
Fig. 1 Concept of operational specifications

れている．宣言部ではスキーマで使用する変数の宣言を行い，述語部ではその変数の制約を記述する．上の例では，宣言部において変数 *member* の宣言を行っている．$\mathbb{P}$ は集合を表し，*member* は人の名前の集合を型とする変数である．また，述語部では会員の数が100人未満であることを表現している．

最後に，会員を登録する操作の仕様を定義する．操作スキーマの構成は状態スキーマと同じである．操作スキーマでは，宣言部で入力，出力および実行前後のシステム内部状態の変数を宣言し，それらの関係を述語部で定義することによりその振舞いを定義する．操作スキーマを集合の概念でとらえると，図1に示すように，入力とシステム前状態の組と，出力とシステム後状態の組との間の関係を定義することに相当する．

会員の登録を表す操作スキーマを以下に示す．

AddPerson

$member, member' : \mathbb{P}\ PERSON$

$p? : PERSON$

$member' = member \cup \{p?\}$

Zでは，変数名の最後に′がついた変数は操作の実行後の変数を表し，′がついていない変数は実行前の変数を表す．上記例では，宣言部においてシステムの前状態 *member*，後状態 $member'$ および入力として $p?$ を宣言している．述語部ではこれらの変数の間の関係を定義している．

2.2 仕様記述変換の概念

仕様記述の作業とは「対象システムの概念をモデル化し，そのモデルを用いてシステムの性質を記述すること」である．対象システムのモデル化を行うためには，対象システムが何であるかを理解することが最も重要であり，対象システムの概念を完全に理解した後で，システムの仕様記述を行うことが理想的である．しかし，システムの概念を完全に理解することは困難な作業であり，システムに対する理解は仕様記述の過程を通して徐々に形成されると考えられる．

本研究では，仕様記述の過程を

(1) 対象システムに対する理解

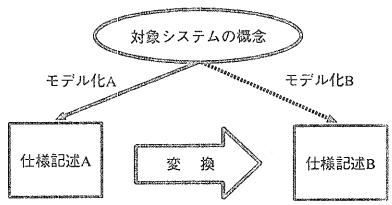

図2 仕様記述変換の概念
Fig. 2 Concept of specification transformations

(2) 最適なモデルの形成

(3) システムに求められる性質の記述

(4) システムの分析

の繰り返しであるとみなしている．仕様記述者は実際に仕様の記述を行うことにより，システムの分析を暗黙のうちに行い対象システムに対する理解が深まる．仕様記述においてシステムの性質を表現する方法はモデルに依存する．より最適な仕様記述を得るためには，対象システムのモデル化が最も重要である．仕様記述の工程では，対象システムの分析を基にして，より最適な仕様記述が試行錯誤で得られると考えられる．

図2は仕様記述変換の概念を図示したものである．仕様記述者は，まず対象システムをモデルAを用いて仕様記述Aを作成した（モデル化A）．仕様記述者は，仕様記述Aを用いてシステムの分析を行いシステムに対する理解が深まり，より最適なモデルBを用いて仕様記述を再構築し仕様記述Bを作成した（モデル化B）．ところで，モデル化Bは対象システムの概念から直接モデル化されたのではなく，モデル化Aで得られた情報を基にして，モデルAからより最適なモデルBへと変換したとみなすことができる．モデル化Aの実線の矢印は，仕様記述Aが対象システムの概念から直接モデル化されたことを表し，モデル化Bの破線の矢印は，仕様記述Bは概念から直接モデル化されたのではなく，仕様記述Aで得られた情報から再構築されたことを表している．本研究では，この再構築の過程を仕様記述変換とみなし，モデルの変更に伴う仕様記述の変換手法を提示することにより，仕様記述におけるモデル化の支援を図る．

3. 2項関係モデルにおける仕様記述変換手法

本研究では，システムの内部状態を表現するための基本的なモデルとして2項関係モデルに着目した．本章では，2項関係モデルにおける仕様記述変換手法の概要を述べた後，具体的な変換手法について説明する．

3.1 概 要

本変換手法では，Zにおける2項関係モデルの中で

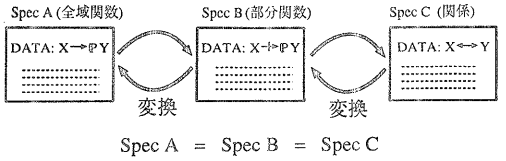

図3 2項関係モデルにおける変換
Fig. 3 Transformations of binary relation model

最も特徴的なモデルである，全域関数，部分関数および関係を変換の対象とする．全域関数は，定義域において未定義項を考慮する必要がないので，仕様記述が扱いやすく，対象システムの概念の本質を端的に表すことができる．しかし，実装を考慮して，定義域を有限の情報として扱うために，部分関数によるモデル化が現実的なモデルとして有効である場合が多い．また，関係による仕様記述では，リレーショナルデータベースにおける表に対して最も親和性の高い記述となっている．

全域関数，部分関数，および関係の間で仕様記述の内容を保持したまま仕様記述を変換することができれば，まず，全域関数のもとで未定義項の関数適用を特別扱いせずに考え易いモデルで仕様記述を検討した後，仕様記述変換を用いて部分関数化さらに関係化という過程をとることにより，プログラムに近い仕様記述を系統的に構築することができる．

また，仕様を修正する場合は，対象システムの概念の本質を端的に現している全域関数の仕様記述で仕様の修正を行うことが望ましいと考えられる．仕様記述者は，仕様の修正を考えやすいモデルを用いて，仕様の修正に本質的な作業に集中することができる．仕様の修正に適したモデルで検討し，プログラムに近い仕様記述は形式的な変換で得ることにより，仕様の修正に伴うエラーの混入を防ぐことが期待できる．

2項関係モデルの変換手法として，

(1) 全域関数と部分関数間の変換手法，および
(2) 部分関数と関係間の変換手法

をそれぞれ提示する (図3)．本変換技術の特徴は，「仕様記述の内容を維持しつつモデルを相互に変換する手法を与えること」である．

全域関数と部分関数間の変換では，全域関数でモデル化されたデータと，部分関数でモデル化されたデータの間の変換について議論する．本手法では，部分関数による表現と全域関数による表現を同じ表現にするために，補助関数を導入して部分関数の定義域に属さない要素の対応を表現する．全域関数と部分関数間の変換は，全域関数，部分関数および補助関数間の関係を明示することにより実現される．

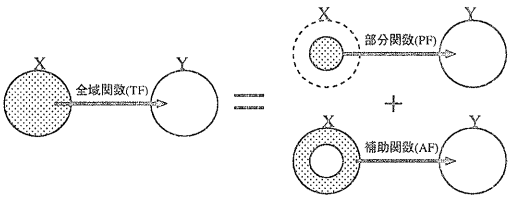

図4 補助関数の導入による全域関数と部分関数の同一視
Fig. 4 Identifying partial functions with total functions by introducing auxiliary functions

部分関数と関係間の変換では，集合 X から 集合 Y の巾集合への部分関数 $X \pfun \mathbb{P}\, Y$ でモデル化されたデータと，関係 $X \rightleftarrows Y$ でモデル化されたデータの間の変換について議論する．本手法では，$X \pfun \mathbb{P}\, Y$ と $X \rightleftarrows Y$ を同一視する概念を導入し，これらの間のデータ構造を変換する関数を定義する．$X \pfun \mathbb{P}\, Y$ と $X \rightleftarrows Y$ 間の変換はこの変換関数を用いて実現される．

3.2 全域関数と部分関数間の変換

本節では，全域関数と部分関数間の変換について述べる．

3.2.1 基本アイデア

全域関数と部分関数間の変換の基本アイデアは，

- 部分関数の定義域に属さない要素に対する補助関数を導入して，全域関数が有する情報と部分関数が有する情報を同一視すること

である．図4では，補助関数（AF）を導入して，全域関数（TF）と部分関数（PF）を同一視する概念を示している．AF，TF，および PF の間には次の条件を満足する必要がある．

$TPACondition[X, Y]$

$tf : X \longrightarrow Y$

$pf, af : X \pfun Y$

$pf = \mathrm{dom}(pf) \lhd tf$

$af = \mathrm{dom}(pf) \lhd\!\!\!\!- tf$

上記のスキーマ $TPACondition$ は関数間の満たすべき条件を記述している．宣言部では全域関数 tf，部分関数 pf および補助関数 af の宣言を行っている．述語部において，1行目の制約

$$pf = \mathrm{dom}(pf) \lhd tf$$

は「pf の対応関係は，pf の定義域に属している tf の対応関係と同じであること」を表現している．また，2行めの制約

$$af = \mathrm{dom}(pf) \lhd\!\!\!\!- tf$$

は「af の対応関係は，pf の定義域に属していない tf

の対応関係と同じであること」を表現している．

以上の関係を満足する部分関数と補助関数を定義すれば，全域関数が有する情報と部分関数が有する情報を同一視することができる．この関係を定義すれば，全域関数と部分関数の間で互いにデータのモデルを変換することが可能となる．ところで，補助関数を定義することは，部分関数のモデル化において定義域に属さない要素の意味を明示することに相当する．

3.2.2 変換手法

本項では，全域関数と部分関数間の変換手法について述べる．変換は次の手順でおこなう．

(1) 補助関数の定義と，変換の対象となる全域関数，部分関数および補助関数の関係をスキーマで記述する．

(2) スキーマ合成を用いて部分関数による仕様記述を得る．

手順の詳細を以下に示す．

関数間のスキーマの定義

補助関数の定義および関数間の関係をスキーマを用いて記述する．

TPARelation

$$tf : X \longrightarrow Y$$
$$pf, af : X \rightarrow\!\!\!\!\!\!+\;\; Y$$

$$tf = pf \cup af$$
$$Predicate$$

宣言部では全域関数 tf，部分関数 pfおよび補助関数 af の宣言を行う．述語部では，関数間の関係と補助関数 af の定義を行う．1行目の制約

$$tf = pf \cup af$$

は，関数間の関係を表現している．2 行目の制約 *Predicate* において，関数間の条件 *TPACondition* を満足する補助関数 af の定義を行う．*Predicate* では，条件 *TPACondition* を満足するように af を定義する必要がある．

部分関数による仕様記述の獲得

全域関数で定義された仕様記述から，部分関数による仕様記述を得る方法について述べる．ここでは，全域関数による仕様記述 *TSpec* に対する変換について考える．*TSpec* は以下の通りである．

TSpec

$$tf : X \longrightarrow Y$$
$$Decl$$

$$Predicate$$

Decl は変換の対象となる変数 tf 以外の変数の宣言を表し，*Predicate* は述語を表す．部分関数による仕様記述 *PSpec* は以下の定義により得られる．

$$PSpec \mathrel{\widehat=} (TSpec \wedge TPARelation) \setminus (tf, af)$$

上の定義では，スキーマ *PSpec* は，

(1) 全域関数によるスキーマ *TSpec* と関数間のスキーマ *TPARelation* を論理積によるスキーマ合成により結び付け，

(2) 全域関数の変数 tf と補助関数の変数 af を変数を隠蔽する演算子（$\setminus$）を用いて宣言から取り除く

ことを意味している．したがって *PSpec* の宣言部は

PSpecDecl

$$pf : X \rightarrow\!\!\!\!\!\!+\;\; Y$$
$$Decl$$

となり，*PSpec* は部分関数を用いた仕様記述となる．

以上で，全域関数から部分関数への変換の手法を示した．逆に，部分関数から全域関数への変換は，同様にして

$$TSpec \mathrel{\widehat=} (PSpec \wedge TPARelation) \setminus (pf, af)$$

により，全域関数による仕様記述 *TSpec* を，部分関数の仕様記述 *PSpec* から得ることができる．

3.3 部分関数と関係間の変換

本節では，部分関数と関係間の変換について述べる．ここでは，集合 X から 集合 Y の巾集合への部分関数 $X \rightarrow\!\!\!\!\!\!+\;\; \mathbb{P}\, Y$ と関係 $X \leftrightarrow Y$ の間の変換について考察する．

3.3.1 基本アイデア

部分関数 $X \rightarrow\!\!\!\!\!\!+\;\; \mathbb{P}\, Y$ と関係 $X \leftrightarrow Y$ 間の変換の基本アイデアは，

- 部分関数 $X \rightarrow\!\!\!\!\!\!+\;\; \mathbb{P}\, Y$ と関係 $X \leftrightarrow Y$ の表記内容を同一視すること

である．これは，例えば集合 X, Y をそれぞれ

$$X = \{a, b, c\}, Y = \{s, t, u\}$$

としたとき，図5に示すように，部分関数 $X \rightarrow\!\!\!\!\!\!+\;\; \mathbb{P}\, Y$ の集合表記

$$\{(a, \{s\}), (b, \{s, t, u\})\}$$

と，関係 $X \leftrightarrow Y$ の集合表記

$$\{(a, s), (b, s), (b, t), (b, u)\}$$

の内容を同一視することである．

これは，部分関数の視点からみれば，「X から Y の集合への部分関数を X と Y の関係として解釈すること」に相当する．逆に，関係の視点からみれば，「X と Y の関係を X から Y の集合への部分関数として解釈すること」に相当する．以上のような部分関数と

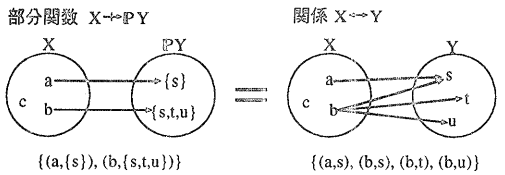

図5 部分関数と関係の内容の同一視
Fig. 5 Identifying relations with partial functions

関係の間の解釈は，以下のスキーマで表現することができる．

$$
\begin{array}{|l}
\hline
IdentifyPR[X, Y] \\
pf : X \pfun \mathbb{P}\, Y \\
rl : X \rel Y \\
\hline
pf = \{x : \mathrm{dom}(rl) \bullet x \mapsto rl(\!|\{x\}|\!)\} \\
rl = \bigcup \{x : \mathrm{dom}(pf) \bullet \{y : pf(x) \bullet x \mapsto y\}\} \\
\hline
\end{array}
$$

部分関数 pf は関係 rl を用いて表現することが可能で，逆に関係 rl も部分関数を用いて表現することが可能である．$IdentifyPR$ の関係からデータ構造を変換する関数を定義することは可能である．この変換関数を用いることにより，どちらのデータ構造にでも仕様記述を変換することができる．変換関数および変換方法については次節で述べる．

3.3.2 変換手法

部分関数 $X \pfun \mathbb{P}\, Y$ と関係 $X \rel Y$ 間の変換手法について述べる．部分関数から関係への変換関数 $P2R$ および関係から部分関数への変換関数 $R2P$ を定義し，仕様記述の変換の方法を示す．

変換関数

変換関数 $P2R$ および $R2P$ を定義する．変換関数 $P2R$ は，部分関数 $X \pfun \mathbb{P}\, Y$ のデータ構造を関係 $X \rel Y$ のデータ構造に変換し，逆に，$R2P$ は，関係 $X \rel Y$ のデータ構造を部分関数 $X \pfun \mathbb{P}\, Y$ のデータ構造に変換する関数である．変換関数は 1 対 1 の関数（単射）として以下のように定義される．

$$
\begin{array}{|l}
\hline
[X, Y] \\
P2R : (X \pfun \mathbb{P}\, Y) \inj (X \rel Y) \\
R2P : (X \rel Y) \inj (X \pfun \mathbb{P}\, Y) \\
\hline
\forall\, pf : X \pfun \mathbb{P}\, Y \bullet \\
\quad P2R(pf) = \bigcup \{x : \mathrm{dom}(pf) \bullet \\
\qquad\qquad \{y : pf(x) \bullet x \mapsto y\}\} \\
\forall\, rl : X \rel Y \bullet \\
\quad R2P(rl) = \{x : \mathrm{dom}(rl) \bullet x \mapsto rl(\!|\{x\}|\!)\} \\
\hline
\end{array}
$$

図5で示した表記例に対して，この変換関数を適用すれば，

$$P2R(\{(a, \{s\}), (b, \{s, t, u\})\}) = \{(a, s), (b, s), (b, t), (b, u)\}$$

$$R2P(\{(a, s), (b, s), (b, t), (b, u)\}) = \{(a, \{s\}), (b, \{s, t, u\})\}$$

を得る．このようにして，変換関数を用いて部分関数と関係の間のデータ構造を自由に換えることができる．

仕様記述の変換

まず，部分関数から関係への仕様記述の変換について考える．この変換では，部分関数の「変数」のデータ構造を関係のデータ構造に変換する．もちろん，変換前の仕様記述の述語部では，「変数」は部分関数として記述されているので，変換後の述語部に変換前と同じ記述を使うことはできない．しかし，関係のデータの意味を部分関数と解釈して，「変数」を部分関数に変換して制約を記述することは可能である．これは，関係の「変数」に対して $R2P$ を適用することにより実現することができる．したがって，変換前と同じ仕様記述を得るには，$R2P$ を用いて変換前の述語部と同じ制約を記述すればよい．これは，変換前の述語部に対して，対象となる変数に $R2P$ を適用することにより得ることができる．

以上の議論より，部分関数 $X \pfun \mathbb{P}\, Y$ から関係 $X \rel Y$ の仕様記述は，以下の手順で得ることができる．

(1) 変換の対象となる変数の型を $X \pfun \mathbb{P}\, Y$ から $X \rel Y$ に換える．
(2) 変換の対象となる変数に $R2P$ を適用する．

以上で，部分関数 $X \pfun \mathbb{P}\, Y$ から関係 $X \rel Y$ への変換手法を示した．逆に，関係から部分関数の仕様記述は，以下の手順で得ることができる．

(1) 変換の対象となる変数の型を $X \rel Y$ から $X \pfun \mathbb{P}\, Y$ に換える．
(2) 変換の対象となる変数に $P2R$ を適用する，

次章では，具体例を用いて仕様記述変換の適用例を示す．

4. 適用例

本章では，具体例として予約管理システムの仕様記述を用いて仕様記述変換の適用例を示す．

4.1 予約管理システム

予約管理システムとは「ある共有施設の予約管理をおこなうためのシステム」である．本システムは，図6で示されるように，予約に関するデータを記録するデータベースを保持し，システムの使用者に対して予約の登録，削除，表示などのサービスを提供する．

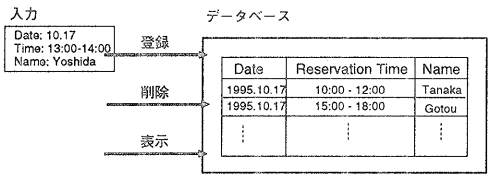

図6 予約管理システム
Fig. 6 Reservation management system

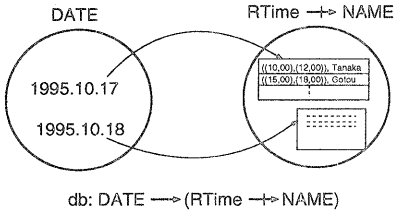

図7 予約管理システムの内部状態のモデル化
Fig. 7 Model of internal states of reservation management system

本論文では操作の例として予約の登録について取り扱う.

4.1.1 Z仕様記述

まず, データの定義を以下に示す.

$[DATE, NAME]$

$Hour == 0\,..\,23;\ Min == 0\,..\,59$

Time

$h : Hour$

$m : Min$

$\_ \mathsf{LE} \_ : Time \leftrightarrow Time$

$\forall x, y : Time \bullet$

$x \mathsf{LE}\ y \Leftrightarrow x.h < y.h \lor (x.h = y.h \land x.m \leq y.m)$

$RTime == \{st, et : Time \mid st \mathsf{LE}\ et\}$

$RTimeSet == \{r : \mathbb{P}\, RTime; xs, xe, ys, ye : Time \mid (xs, xe) \in r \land (ys, ye) \in r \land (xe \mathsf{LE}\ ys \lor ye \mathsf{LE}\ xs) \bullet r\}$

次に予約管理システムの内部状態を定義する. システムの内部状態は, 予約に関するデータを保持するデータベースに相当する. 予約状態を管理するデータベースのモデル化はいろいろ考えられるが, ここでは図7に示すように. 日付の集合を定義域, 予約時間から名前への関数を値域とする全域関数としてモデル化した. システムの内部状態および初期状態を示すスキーマを以下に示す.

State

$tf : DATE \longrightarrow (RTime \nrightarrow NAME)$

$\forall d : \mathrm{dom}(tf) \bullet \mathrm{dom}(tf(d)) \in RTimeSet$

InitState

State

$\forall d : \mathrm{dom}(tf) \bullet tf(d) = \varnothing$

最後に予約登録の操作を以下に示す.

Entry

$\Delta State$

$d? : DATE;\ st?, et? : Time;\ n? : NAME$

$(st?, et?) \in RTime$

$(st?, et?) \notin \mathrm{dom}(tf(d?))$

$tf' = tf \oplus \{d? \mapsto (tf(d?) \cup \{(st?, et?) \mapsto n?\})\}$

宣言部における $\Delta State$ は, 操作を実行する時の *State* の前状態と後状態の変数の宣言を表す. データベースの登録は,

$tf' = tf \oplus \{d? \mapsto (tf(d?) \cup \{(st?, et?) \mapsto n?\})\}$

の述語で表現されている.

4.1.2 全域関数から部分関数への変換

本項では全域関数から部分関数への変換の適用を試みる. 4.1.1項では, システム内部状態を表すスキーマ *State* において, データベースの構造を全域関数

$tf : DATE \longrightarrow (RTime \nrightarrow NAME)$

としてモデル化した. ここでは, これを部分関数

$pf : DATE \nrightarrow (RTime \nrightarrow NAME)$

としてモデル化した仕様記述に変換する. まず, 全域関数および部分関数によるモデル化についての考察をおこない, 関数間のスキーマを定義する. その後, 全域関数による仕様記述から部分関数による仕様記述を得る.

全域関数では, 全ての日付に対応する予約情報をデータベースとして保持するという概念に基づいてモデル化されている. これを部分関数を用いてモデル化するためにはどのような補助関数を導入すればよいだろうか. ここでは,

- 部分関数で保持していない日付に対しては「予約がない」とみなす

というモデル化を考える.「予約がない」は, 空集合を対応させることで表現することができる. 補助関数として以下の af を導入する.

$af = \{d : DATE \mid d \notin \mathrm{dom}(pf) \bullet d \mapsto \varnothing\}$

関数間のスキーマを以下のように定義する．

TPARel

$$State$$
$$pf, af : DATE \pfun (RTime \pfun NAME)$$
$$tf = pf \cup af$$
$$af = \{d : DATE \mid d \notin \mathrm{dom}(pf) \bullet d \mapsto \emptyset\}$$

上の定義では，pf と af の定義域は排他的である．つまり定義域の要素に重なりはないので，関数間の条件 *TPACondition* の制約

$$pf = \mathrm{dom}(pf) \ndres tf \land af = \mathrm{dom}(pf) \nrres tf$$

は確かに満足されている．

次に，*TPARel* を用いて部分関数による仕様記述を得る．全域関数による仕様記述 *State*, *InitState*, *Entry* に対して部分関数による仕様記述 *PState*, *PInitState*, *PEntry* を定義する．

$$PState \mathrel{\widehat=} (State \land TPARel) \setminus (tf, af)$$
$$PInitState \mathrel{\widehat=} (InitState \land TPARel) \setminus (tf, af)$$
$$PEntry \mathrel{\widehat=} (Entry \land TPARel \land TPARel') \setminus (tf, af, tf', af')$$

上で定義したスキーマを展開し簡単化した仕様記述を以下に示す．

ExpandPState

$$pf : DATE \pfun (RTime \pfun NAME)$$
$$\forall d : \mathrm{dom}(pf) \bullet \mathrm{dom}(pf(d)) \in RTimeSet$$

ExpandPInitState

$$PState$$
$$pf = \emptyset$$

ExpandPEntry

$$\Delta PState$$
$$d? : DATE;\ st?, et? : Time;\ n? : NAME$$
$$(st?, et?) \in RTime$$
$$d? \notin \mathrm{dom}(pf) \land pf' = pf \cup \{d? \mapsto \{(st?, et?) \mapsto n?\}\} \lor$$
$$d? \in \mathrm{dom}(pf) \land (st?, et?) \notin \mathrm{dom}(pf(d?)) \land pf' = pf \oplus \{d? \mapsto (pf(d?) \cup \{(st?, et?) \mapsto n?\})\}$$

以上のようにして，全域関数による仕様記述 *State*, *InitState*, *Entry* から部分関数による仕様記述 *PState*, *PInitState*, *PEntry* を変換により得ることができた．

4.1.3 部分関数から関係への変換

本項では部分関数から関係への変換の適用を試みる．変換は部分関数による仕様記述 *PState*, *PInitState*, *PEntry* に対して行う．

4.1.2項では，データベースのデータ構造を

$$pf : DATE \pfun (RTime \pfun NAME)$$

としてモデル化した．Zでは，関数は直積の集合として捉えているので，pf は，

$$pf : DATE \pfun \mathbb{P}(RTime \times NAME)$$

と記述することができる．ただし，pf には関数の制約

$$\forall r : \mathrm{ran}(pf) \bullet r \in RTime \pfun NAME$$

を付け加える必要がある．したがって，4.1.2項における，システムの内部状態を表すスキーマ *PState* は，以下のように展開することができる．

PStateAlt

$$pf : DATE \pfun \mathbb{P}(RTime \times NAME)$$
$$\forall r : \mathrm{ran}(pf) \bullet r \in RTime \pfun NAME$$
$$\forall d : \mathrm{dom}(pf) \bullet \mathrm{dom}(pf(d)) \in RTimeSet$$

上のスキーマ *PStateAlt* に対して変換を適用すれば，関係による内部状態のスキーマ *RState* を得る．

RState

$$rl : DATE \rightleftarrow (RTime \times NAME)$$
$$\forall r : \mathrm{ran}(R2P(rl)) \bullet r \in RTime \pfun NAME$$
$$\forall d : \mathrm{dom}(R2P(rl)) \bullet \mathrm{dom}(R2P(rl)(d)) \in RTimeSet$$

RState は以下のように簡単化することができる．

ExpandRState

$$rl : DATE \rightleftarrow (RTime \times NAME)$$
$$\forall r : \{d : \mathrm{dom}(rl) \bullet rl(\!|\{d\}|\!)\} \bullet r \in RTime \pfun NAME$$
$$\forall d : \mathrm{dom}(rl) \bullet \mathrm{dom}(rl(\!|\{d\}|\!)) \in RTimeSet$$

同様にして，*PInitState*, *PEntry* に対しても変換を適用すると，関係による仕様記述 *RInitState*, *REntry* を得る．

RInitState

$$RState$$
$$R2P(rl) = \emptyset$$

REntry

$\Delta RState$

$d? : DATE;\ st?, et? : Time;\ n? : NAME$

$(st?, et?) \in RTime$

$d? \notin \mathrm{dom}(R2P(rl)) \land$

$R2P(rl') = R2P(rl) \cup$

$\{d? \mapsto \{(st?, et?) \mapsto n?\}\} \lor$

$d? \in \mathrm{dom}(R2P(rl)) \land$

$(st?, et?) \notin \mathrm{dom}(R2P(rl)(d?)) \land$

$R2P(rl') = R2P(rl) \oplus$

$\{d? \mapsto (R2P(rl)(d?) \cup$

$\{(st?, et?) \mapsto n?\})\}$

変換により得られた *RInitState*, *REntry* は，以下のように簡単化することができる．

ExpandRInitState

RState

$rl = \emptyset$

ExpandREntry

$\Delta RState$

$d? : DATE;\ st?, et? : Time;\ n? : NAME$

$(st?, et?) \in RTime$

$(st?, et?) \notin \mathrm{dom}(rl(\!|\{d?\}|\!))$

$rl' = rl \cup \{d? \mapsto ((st?, et?), n?)\}$

以上のようにして，部分関数による仕様記述 *PState*, *PInitState*, *PEntry* から関係による仕様記述 *RState*, *RInitState*, *REntry* を変換により得ることができた．

5. 考　　察

本章では，形式的手法による従来の開発手法との比較を行った後，2項関係モデルにおける変換の特徴と，ソフトウェア開発における変換手法の意義について議論する．

5.1 形式的手法による従来の開発手法との比較

Zは現存している形式仕様記述言語の中で最も利用されている言語の一つであり，Zを用いた様々な事例報告は文献9)において多数紹介されている．Zを用いたソフトウェア開発手法として，抽象度の高い仕様記述からプログラムコードと同等のレベルまで，段階的詳細化によるトップダウンの開発手法が提案されている[5),11)]．また，表記はZを用い，仕様記述のレベル間の保証は Refinement Calculus[12)]を用いた統合的な手法[13)14)]も提案されている．これらの手法は，最も抽象度の高い記述から，データ詳細化およびアルゴリズム詳細化といった技術をもとに，系統的にインプリメントに近づける際に有効な手法である．しかし，仕様記述のもとになる最も抽象度の高い記述をどのようにして得たかについては，そこには述べられていない．仕様記述の過程では，対象システムをどのようにモデル化するかが最も重要な作業であり，このモデル化の作業をいかにして支援するかがソフトウェア開発における重要な課題である．

本研究では，対象システムの最適なモデルは仕様記述の過程において試行錯誤の結果確定することに着目し，仕様記述の過程を変換と捉えその過程を定式化することにより，仕様記述におけるモデル化作業の支援を図った．システムの最適なモデルは最初から得られるのではなく，システムの記述が進むにつれて徐々にシステムに対する理解が形成され，より良いモデルを用いて仕様記述が書き換えられる．本研究の基本的なアイデアは，この書き換えの過程を解明し変換手法として提示することにより，モデルの変更に伴う記述の変換を系統的に行う手段を与えることである．仕様記述段階での最適なモデルを系統的に模索する方法の提示は，従来，仕様記述者がアドホックに行っていたモデル化の模索を改善する意義があると考えられる．

5.2 2項関係モデルにおける変換の特徴

本変換技術の特徴は，「仕様記述の内容を維持しつつモデルを相互に変換する手法を与えること」である．本論文では，2項関係モデルにおける変換手法として，(1) 全域関数と部分関数間の変換手法と，(2) 部分関数と関係間の変換手法を提示した．以降では，それぞれの変換手法の特徴について述べる．

5.2.1 全域関数と部分関数間の変換手法の特徴

全域関数と部分関数間の変換手法では，部分関数の定義域に属さない要素に対する補助関数を定義することにより，全域関数と部分関数による仕様記述間の形式的な相互変換を可能にした．ここで，形式的な変換とは，仕様の意味を解釈することなく仕様記述を記号処理として変換することを言う．補助関数を定義することは，全域関数と部分関数が有する情報を同一視するために，仕様の意味を解釈して部分関数のモデル化における定義域に属さない要素に対する対応関係を明示したことを意味する．したがって，全域関数と部分関数間の変換では，補助関数を定義するために仕様の意味を解釈する必要がある．しかし逆に言えば，補助関数を定義さえすれば，全域関数と部分関数の間で相互に変換が可能である．

ところで，部分関数の観点から補助関数を定義する

ことの意義について考えると，定義域に属さない要素の対応関係を定義することは，仕様記述には明示されない暗黙の関係を陽に記述することを仕様記述者に対して強制する．ソフトウェア開発では，仕様記述の段階における開発者間での誤解が後のソフトウェア開発に大きな影響を与える．開発者間における誤解の原因の一つとして，一見自明と思われる仕様の意味が明記されていないことが考えられる．開発者間で対象システムに対する共通の認識を形成するためは，仕様はできるだけ明記されていることが望ましい．補助関数を定義することは，仕様の誤解を少なくするための暗黙の仕様を明示することに効果がある．

5.2.2 部分関数と関係間の変換手法の特徴

部分関数と関係間の変換手法では，集合 X から集合 Y の巾集合への部分関数 $(X \nrightarrow \mathbb{P}\ Y)$ と，集合 X と集合 Y の関係 $(X \leftrightarrow Y)$ に関して，部分関数と関係の間のデータ構造に関する変換関数を提示することにより，部分関数と関係による仕様記述間の形式的な相互変換を可能にした．したがって，部分関数と関係間の変換では，仕様の意味を解釈することなく，仕様記述を記号処理として変換することができる．

本変換における部分関数から関係への変換は，リレーショナルデータモデルにおける正規化[8] に相当すると捉えることができる．ここでいう正規化とは，レコードのフィールド値として集合の値を許す非正規形リレーションを，フィールド値に単一の値をとる第一正規形 (first normal form) に直すことを言う．リレーショナルデータベースにおける正規化では表に関する変換を提示しているが，本変換では操作の仕様記述含めた仕様記述全体の変換を可能にしている．ところで，関係の仕様記述の観点からデータ構造に関する変換関数の意義について考えると，データ構造に関する変換関数は，関係のデータをどのように解釈するかを明示することに相当すると考えられる．

5.3 ソフトウェア開発における変換手法の意義

ソフトウェア開発における2項関係モデルにおける変換手法の意義について，仕様記述の過程と仕様の修正作業の過程の観点から考察する．

5.3.1 仕様記述の過程

本論文の適用例で示した，全域関数，部分関数，関数への仕様記述の流れは，仕様記述における一つの指針となり得ると考えられる．全域関数によるモデル化では，定義域において未定義項を考慮する必要がないので，仕様記述が扱いやすく，対象システムの概念の本質を端的に表しており，システムの本質を理解するには最も適したモデルであると考えられる．しかし，2項関係の情報をシステムの内部状態として保持するためには，定義域として有限の情報を扱うために部分関数によるモデル化が現実的なモデルとして有効である場合が多い．関係による仕様記述では，リレーショナルデータベースにおける表に対して最も親和性の高い記述となっている．

ところで，部分関数によるモデル化では，定義域における未定義項の関数適用を考慮する必要があるので仕様記述は複雑になる傾向がある．そこで，まず，全域関数のもとで未定義項の関数適用を特別扱いせずに考え易いモデルで仕様記述を検討した後，仕様記述変換を用いて部分関数化さらに関係化という過程をとることにより，プログラムに近い仕様記述を系統的に構築することができる．

5.3.2 仕様の修正作業の過程

システムの仕様を修正する場合は，システムの仕様の概念が最も明確に現れている仕様記述で検討することが望ましい．関係の仕様記述はプログラムに近い仕様記述であり，また，部分関数による仕様記述は先に述べたように定義域における未定義項の関数適用を考慮する必要がある．したがって，対象システムの概念の本質を端的に現している全域関数の仕様記述で仕様の修正を行い，その後，部分関数化さらに関係化という過程が有効な手段であると考えられる．

仕様の修正作業を行うときには，全域関数と部分関数の変換における補助関数は既に定義されているので，全域関数で仕様を修正した後の変換作業は，全て記号処理として変換を取り扱うことができる．仕様記述者は，仕様の修正を考えやすいモデルを用いて，仕様の修正に本質的な作業に集中することができる．仕様の修正に適したモデルで検討し，プログラムに近い仕様記述は形式的な変換で得ることにより，仕様の修正に伴うエラーの混入を防ぐことが期待できる．

5.4 関連技術

システム仕様記述は，基本的には，システムが保持すべき内部状態と，システムが必要な機能を記述することである．より良いシステム記述を構築するためには，システムの内部状態のモデル化と仕様記述の構造が重要である．また，システムの内部状態が保持すべき制約を明確に記述することが，システム分析において重要な役割を果たす．更に，モデル化の作業を支援するために，直観的で人間になじみやすい図式表現の導入が望まれる．

筆者らは仕様記述変換技術として，

- 機能分割手法[15]
- 操作仕様記述からのシステム状態不変条件の抽

出[16]
- 図式表現から形式的な表現への変換[17]

の研究を行っている．

「機能分割手法」では，機能分割をZのスキーマ分割として機能の内容を変えずに記号上の操作による分割を行う方法を提案している．大規模ソフトウェア開発においては，機能分割は重要な課題の一つであり，本手法では系統的に機能分割を行う方法を提示している．「操作仕様記述からのシステム状態不変条件の抽出」では，操作の制約がシステム状態不変条件に及ぼす影響を分析し，操作の記述からシステム状態不変条件を抽出する方法を提案している．操作の仕様記述として局所的に表現されている制約を，システム状態という大局的な制約として変換する手法を提示している．また，「図式表現から形式的な表現への変換」では，形式的な仕様記述の作成を支援するために，図式表現からZ仕様記述に変換する手法を提案している．図式表現から形式的な記述を得ることにより，形式的な仕様記述の上でダイアグラム間の一貫性の検証を行うことを可能にしている．これらの変換技術を基にして，ソフトウェア開発手法を体系化することが今後の課題である．

6. おわりに

本論文では，形式的手法による仕様記述変換技術として，2項関係モデルにおける仕様記述変換の手法を提示した．Zにおいて2項関係モデルの中で最も特徴的なモデルである，全域関数（$\longrightarrow$），部分関数（$\nrightarrow$）および関係（$\leftrightarrow$）の間の対応関係を分析し，

(1) 全域関数と部分関数間の変換手法，および
(2) 部分関数と関係間の変換手法

を提示した．

本変換技術の特徴は，「仕様記述の内容を維持しつつモデルを相互に変換する手法を与えること」である．変換技術の意義として，

- モデル変換に伴う暗黙の関係の明示化
- 記述を書き換えるための形式的な作業と意味に関わる作業の分離

があげられる．変換するモデル間の対応関係を明示することは，仕様記述者が暗黙のうちに対応づけているモデル間の違いの一面を特徴づけることに相当する．モデル間の相違を特徴づけることにより，モデルを変更する仕様記述者の意図を明確にし，仕様記述として陽に表現されない暗黙の関係を明らかにする．また，仕様記述変換技術は，ソフトウェア開発における意義として，

- 仕様記述を構築する時の指針の提供，
- 仕様の検討および検証をする際の目的に応じた最適なモデルの選択，

を可能にすることにより，仕様記述に対する信頼性向上に寄与できると考えられる．今後の課題としては，

- 同等な制約記述の分析による記述変換手法の提示
- 仕様記述変換の機械による支援

が挙げられる．形式的な仕様を記述するためには，論理式による記述は避けられない．同じ制約を表現する論理式の記述を分析し，それらの間の論理式の関係を変換規則として提示することにより，仕様記述の支援を図ることができるであろう．このような変換規則も仕様記述変換技術とみなすことができる．今後，様々な変換規則を開発し，仕様記述の変換を機械で支援する環境を構築することにより，仕様記述の支援を図る予定である．

参考文献

1) D.Craigen,S.Gerhart and T.Ralston: An international survey of industrial applications of formal methods, Volume 1 study methodology, Tech.Report PB93-178556/AS, National Technical Information Service (1993).
2) D.Craigen,S.Gerhart and T.Ralston: An international survey of industrial applications of formal methods, Volume 2 case studies, Tech.Report PB93-178564/AS, National Technical Information Service (1993).
3) A.Hall: Seven myths of formal method, *IEEE Software*, Vol.7,No.5,pp.11-19 (1990).
4) J.M.Spivey: *The Z Notation - a Reference Manual 2nd ed.*, Prentice Hall (1992).
5) J.B.Wordsworth: *Software Development with Z: A Practical Approach to Formal Methods in Software Engineering*, Addison-Wesley (1992).
6) B.Potter,J.Sinclair and D.Till: *An Introduction to Formal Specification and Z*, Prentice Hall (1991).
7) A.Diller: *Z: An Introduction to Formal Methods*, John Wiley & Sons (1990).
8) 増永良文: リレーショナルデータベース入門, サイエンス社 (1991).
9) I.J.Hayes,editor: *Specification Case Studies 2nd ed.*, Prentice Hall (1993).
10) J.Woodcock and J. Davies: *Using Z Specification, Refinement, and Proof*, Prentice Hall (1996).
11) S.King and I.H.Sφrensen: From specification, through design to code:a case study in refinement, formal methods - *Theory and Practice* (ed. P.N.Scharbach), pp.103-137, Blackwell

Scientific Publications (1989).
12) C.Morgan: *Programming from Specifications*, Prentice Hall (1990).
13) S.King: Z and the refinement calculus, *Lecture Notes in Computer Science*, Vol.428, pp.164-188, Springer-Verlag (1990).
14) K.R.Wood: A practical approach to software engineering using z and the refinement calculus, *ACM Software Engineering Notes*, Vol.18, No.5, pp.79-88 (1993).
15) 張漢明，荒木啓二郎: 形式的手法による機能分割手法，第1回ソフトウェア工学の基礎ワークショップ FOSE'94 論文集，pp.129-134 (1994).
16) 張漢明，荒木啓二郎: 操作仕様記述におけるシステム状態不変条件の抽出，ソフトウェア工学の基礎II，レクチャーノート／ソフトウェア学15，pp.183-188，近代科学社 (1996).
17) 河野勝利，張漢明，荒木啓二郎: 形式的手法に基づいた構造化ダイアグラムの一貫性検証について，コンピュータソフトウェア，Vol.15，No.3，pp.2-16 (1998).

(平成10年10月30日受付)
(平成10年12月18日再受付)
(平成11年 1月 8日採録)

張 濱明（正会員）

1963年生．1989年同志社大学工学部卒業．同年より1992年までオムロンソフトウェア(株)勤務．1995年奈良先端科学技術大学院大学情報科学研究科博士前期課程修了．現在，同研究科博士後期課程在学中．(財)九州システム情報技術研究所研究員．形式的仕様記述の研究に従事．日本ソフトウェア科学会会員．

荒木啓二郎（正会員）

1954年生．1976年九州大学工学部卒業．1978年同大学院修士課程修了．九州大学助手，同助教授，奈良先端技術大学院大学教授を経て，現在，九州大学大学院システム情報科学研究科教授．(財)九州システム情報技術研究所研究室長兼務．工学博士．プログラミング言語，形式的仕様記述，インターネット，マルチメディア通信等の研究に従事．ソフトウェア技術者協会常任幹事，九州地域研究ネットワーク(KARRN)協会事務局長，博多祇園山笠元赤手拭等．